



# natural

---

Version 4.1.2 for Mainframes | XML Toolkit

This document applies to Natural Version 4.1.2 for Mainframes and to all subsequent releases.

Specifications contained herein are subject to change and these changes will be reported in subsequent release notes or new editions.

© Copyright Software AG 1979 - 2003.  
All rights reserved.

The name Software AG and/or all Software AG product names are either trademarks or registered trademarks of Software AG. Other company and product names mentioned herein may be trademarks of their respective owners.

# Table of Contents

<b>Natural XML Toolkit</b>	1
Natural XML Toolkit	1
<b>Natural XML Toolkit - Introduction</b>	2
Natural XML Toolkit - Introduction	2
XML Toolkit on the Mainframe	2
Natural XML Toolkit Features	2
XML Toolkit Description	2
Objective	2
General Architecture	2
Map Natural Data Definitions to DTD	3
Serialize Data to XML	4
Map DTD to Natural Data Definitions	4
Parse XML File and Assign to Natural Data	5
Outlook	5
Limitations	5
Solution	5
<b>Using the XML Toolkit</b>	6
Using the XML Toolkit	6
Prerequisite	6
Natural Version	6
Storage Requirements	6
Schema Support	6
Work File Processing	6
Work File Support	6
Print Files	8
Invoking the Application	8
PF-Key Assignments	9
<b>Setting up Specific Generation Options</b>	10
Setting up Specific Generation Options	10
Invoking the Generation Options Setup Screens	10
First Screen	10
Second Screen	12
<b>Using a Natural Data Area as Data Source</b>	14
Using a Natural Data Area as Data Source	14
Select Natural Data Area	14
Select Root Group	15
Generate Copycode for XML Parser Callback	16
Generate Copycode for Serialization	17
Generate File with DTD Definition	18
Show Generation Results	18
<b>Using a Document Type Definition as Data Source</b>	20
Using a Document Type Definition as Data Source	20
Generate From Document Type Definition	20
Select Root Element	21
Generate Copycode	21
Serialize into XML Document	23
Generate Natural Data Area	24
Show Generation Results	25
<b>Natural Simple XML Parser</b>	26
Natural Simple XML Parser	26
Parser Description and Example	26
Parser Restrictions	32

<b>Examples</b> . . . . .	33
Examples . . . . .	33
Serialize Copycode . . . . .	33
Generated Natural Data Area . . . . .	37
Natural DTD Parser . . . . .	39
Generated Type Definition . . . . .	39
Parser CALLBACK Copycode . . . . .	40
<b>Parser Error Messages</b> . . . . .	45
Parser Error Messages . . . . .	45

# Natural XML Toolkit

## Preface

XML, the eXtensible Markup Language, has become one of the most relevant standards and a driving force for Web applications. As of Natural Version 4 for mainframes, XML documents can be created, processed and made available. Because complete highly nested XML documents can be very large, Natural's data-definition and data-manipulation capabilities have been extended with large (up to one gigabyte) and dynamic variables.

The XML Toolkit provides Natural with XML document processing capabilities. A Natural data definition can be generated from an XML Document Type Definition (DTD), and vice versa. The content of a Natural variable can be serialized into an XML document. And an XML document can be parsed into a Natural variable.

## This Document

This document describes an example application that demonstrates the use of XML within a Natural-for-Mainframes environment without external program parts. The following topics are covered:

- Natural XML Toolkit - Introduction
- Using the XML Toolkit
- Setting Up Specific Generation Options
- Using a Natural Data Area as Data Source
- Using a Document Type Definition as Data Source
- Natural Simple XML Parser
- Examples
- Parser Error Messages

Specifications contained herein are subject to change and these changes will be reported in subsequent release notes and new editions.

# Natural XML Toolkit - Introduction

The following topics are covered:

- XML Toolkit on the Mainframe
  - Natural XML Toolkit Features
  - XML Toolkit Description
  - Outlook
  - Limitations
- 

## XML Toolkit on the Mainframe

Natural for Mainframes does not support the REQUEST DOCUMENT statement. Therefore, XML documents have to be processed using Natural work file processing.

If, however, a computer running Natural Version 5 on UNIX, OpenVMS or Windows is accessible on the network, the REQUEST DOCUMENT statement can be executed on this remote machine, using the Natural Remote Procedure Call.

## Natural XML Toolkit Features

- Natural-based XML parser using dynamic variables.
- Functions for
  - conversion of Natural data structures into DTD definitions;
  - generation of COMPRESS statements to save a Natural data structure as an XML document;
  - generation of callback for the Natural-based parser.

## XML Toolkit Description

### Objective

The objective of the Natural XML Toolkit is to provide additional XML functionality with Natural and improve the integration of Natural applications with XML without using external software components like "msxml". It shall be understood as an intermediate step before implementing full XML functionality in Natural's language.

### General Architecture

The Natural XML Toolkit consists of a collection of Natural programs. Some of these are delivered in source format. The Toolkit programs may be integrated into customer applications to provide access to XML data or to deliver data from Natural in XML format. Together with the Toolkit programs, Natural 6 includes example programs and Natural dialogs to control the Toolkit. The XML Toolkit and the new statement REQUEST DOCUMENT provide access to any source in the Internet as sole basis for the implementation of applications that use XML in Natural.

The Natural XML Toolkit is implemented as a Natural dialog that calls the functions listed below:

### XML Toolkit Functions

1. Mapping of Natural Data Definition to DTD and vice versa.  
Document type descriptions are most commonly used to describe the structure of a XML document.
2. XML Token => NAT Data

After the Natural data structure has been created, the XML document has to be parsed and saved into the data structure. An implementation for the delivered XML parser callback routine will be generated. This callback assigns the value of a data element to the corresponding data structure.

### 3. NAT Data => XML Document ("Serialize")

Serialization is the process of taking the data stored in the Natural data structures and creating an XML document according to the description provided in the DTD.

A Natural dialog implements the user interface to the XML Toolkit functions. The DTD will be accessed as a work file and the generated Natural objects will be saved directly to the Natural system file.

## Map Natural Data Definitions to DTD

This mapping is the first step to bind Natural data structures to XML tags and is required to implement a representation of Natural data as XML tags. The example below shows the mapping as well as some obvious differences between Natural and a DTD.

### Natural PDA

Press ESC to enter command mode							
Mem:	EMPL	Lib:	SYSEXXT	Type: PARAMETER	Bytes: 1072	Line: 0	of: 26
C	T	Comment					
*		*** Top of Data Area ***					
	1	EMPLOYEE					
	2	ATTRIBUTES_OF_EMPLOYEE					
	3	PERSONNEL-ID		A		8	
*							
	2	FULL-NAME					
	3	FIRST-NAME		A		20	
	3	NAME		A		20	
*							
	2	FULL-ADDRESS					
	3	C@ADDRESS-LINE		I		4	
	3	ADDRESS-LINE		A		20 (1:6)	
	3	CITY		A		20	
	3	ZIP		A		20	
	3	COUNTRY		A		3	
*							
	2	TELEPHONE					
	3	AREA-CODE		A		6	
	3	PHONE		A		15	

### Generated DTD

```
<!ELEMENT EMPLOYEE ( PERSONNEL-ID, FULL-NAME, FULL-ADDRESS, TELEPHONE, INCOME* )>
<!ELEMENT PERSONNEL-ID (#PCDATA ) >
<!ELEMENT FULL-NAME (FIRST-NAME, NAME )>
  <!ELEMENT FIRST-NAME (#PCDATA )>
  <!ELEMENT NAME (#PCDATA )>
<!ELEMENT FULL-ADDRESS (ADDRESS-LINE*, CITY, ZIP, COUNTRY )>
  <!ELEMENT ADDRESS-LINE (#PCDATA )>
  <!ELEMENT CITY (#PCDATA )>
  <!ELEMENT ZIP (#PCDATA )>
  <!ELEMENT COUNTRY (#PCDATA )>
...
```

The generated DTD will be used later on during serialization to a XML document (see below).

## Serialize Data to XML

During execution of a Natural program, the content of the data defined in the DEFINE DATA statement will be filled with "real" content. This content will be written to a dynamic variable in XML format during serialization and will use the formerly generated DTD as input.

The XML Toolkit generates the program to serialize the data.

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<EMPLOYEE PERSONNEL-ID="30016509">
<FULL-NAME>
  <FIRST-NAME>ELSPETH</FIRST-NAME>
  <NAME>TROWBRIDGE</NAME>
</FULL-NAME>
<FULL-ADDRESS>
  <ADDRESS-LINE>91 BACK LANE</ADDRESS-LINE>
  <ADDRESS-LINE>BILSTON</ADDRESS-LINE>
  <ADDRESS-LINE>STAFFORDSHIRE</ADDRESS-LINE>
  <CITY>BILSTON</CITY>
  <ZIP>ST2 3KA</ZIP>
  <COUNTRY>UK</COUNTRY>
</FULL-ADDRESS>
<TELEPHONE>
  <PHONE>863322</PHONE>
  <AREA-CODE>0602</AREA-CODE>
</TELEPHONE>
...
```

## Map DTD to Natural Data Definitions

The mapping of a DTD to Natural data structures again shows differences. The DTD does not specify how many person records will be included in the XML document, therefore the Toolkit assumes that a maximum number of "v" persons will be included. The application programmer might know the exact number and the data structure could be adapted accordingly. A similar limitation exists with the length of the data. The DTD does not include information about the length of the data in a person's record. Therefore the Toolkit creates fields in the data structure with a length of 253, the current maximum.

```
* DTD E:\SAG\nat\6.1.1\fnat\SYSEXXTG\RES\empl.dtd
COMPRESS &1& '<EMPLOYEE'
  '<PERSONNEL-ID="'EMPLOYEE.PERSONNEL-ID "'>' INTO &1& LEAVING NO
/* now the children
COMPRESS &1& '<FULL-NAME'
  '>' INTO &1& LEAVING NO
/* now the children
COMPRESS &1& '<FIRST-NAME'
  '>'
  EMPLOYEE.FIRST-NAME
  '</FIRST-NAME>' INTO &1& LEAVING NO
COMPRESS &1& '<NAME'
  '>'
  EMPLOYEE.NAME
  '</NAME>' INTO &1& LEAVING NO
/*
COMPRESS &1& '</FULL-NAME>' INTO &1& LEAVING NO
COMPRESS &1& '<FULL-ADDRESS'
  '>' INTO &1& LEAVING NO
/* now the children
```

```

FOR &2& = 1 TO EMPLOYEE.C@ADDRESS-LINE
  COMPRESS &1& '<ADDRESS-LINE'
  '>'
  EMPLOYEE.ADDRESS-LINE(&2&)
  '</ADDRESS-LINE>' INTO &1& LEAVING NO
END-FOR
...

```

## Parse XML File and Assign to Natural Data

```

* DTD E:\SAG\nat\6.1.1\fnat\SYSEXXTG\RES\empl.dtd
DECIDE ON FIRST &1&
  VALUE 'EMPLOYEE'
  RESET INITIAL EMPLOYEE
  VALUE 'EMPLOYEE/@PERSONNEL-ID'
  /* #REQUIRED
  EMPLOYEE.PERSONNEL-ID := &3&
  VALUE 'EMPLOYEE/FULL-NAME'
  IGNORE
  VALUE 'EMPLOYEE/FULL-NAME/FIRST-NAME'
  IGNORE
  VALUE 'EMPLOYEE/FULL-NAME/FIRST-NAME/$'
  EMPLOYEE.FIRST-NAME := &3&
  VALUE 'EMPLOYEE/FULL-NAME/NAME'
  IGNORE
  VALUE 'EMPLOYEE/FULL-NAME/NAME/$'
  EMPLOYEE.NAME := &3&
...

```

## Outlook

The XML Toolkit is another step forward to full XML support with Natural. The XML Toolkit might be extended after the first release. Programs to map Natural data to a Tamino Schema are subjects of investigation. However, the main objective is to implement XML functionality in one of the forthcoming releases as part of Natural's powerful language.

## Limitations

Data structures which will result with more than approximately 700 data fields and groups will end up with the message:

```
Input Structure too big
```

The resulting data structure may be too large for a single data structure, because all fields will be generated as A253 strings.

## Solution

Split up the data structure into smaller sections.

# Using the XML Toolkit

The following topics are covered:

- Prerequisite
  - Work File Processing
  - Print Files
  - Invoking the Application
  - PF-Key Assignments
- 

## Prerequisite

### Natural Version

Natural Version 4 for Mainframes is required.

### Storage Requirements

Depending on the size, complexity or the recursion depth of the processed XML DTD, the XML Toolkit might require up to several hundred kilobytes of space in the DATSIZE buffer.

Parsing an XML document using the generated callback routine requires that the entire document is contained in a dynamic variable.

### Schema Support

Schemas are not supported by the XML Toolkit on mainframe computers; only DTDs can be processed.

## Work File Processing

By default, the XML Toolkit uses Natural Work File 12 as input or output file for DTDs and Work File 13 as error logfile.

The sample parser program, delivered with the XML Toolkit, reads the XML document to be parsed from Work File 12 and writes the output of the pars process to Work File 13.

### Work File Support

Running the XML Toolkit under TP monitors which do not support Natural work files is possible using PC work files, if you are working with a PC on which Entire Connection is in use.

XML Documents which shall be parsed using the generated callback routine or which shall be generated using the generated serialization code can be accessed via work files as well. Data must be accessed with a variable of type ALPHA DYNAMIC. The work file should be of type UNFORMATTED.

PC files can be accessed using the subprogram XML2PCWR provided in the SYSEXXT library. XML2PCWR writes or reads an ALPHA DYNAMIC variable to or from a PC work file.

### Calling Conventions for XML2PCWR:

```
CALLNAT 'XML2PCWR' XML-PAGE FILENUMBER OPERATION RETCODE
```

**Parameter Definition:**

```
DEFINE DATA LOCAL
  1 XML-PAGE          (A)      DYNAMIC          /* XML page
  1 FILENUMBER        (I2)
  1 OPERATION         (A2)     /* work file number: 1 - 31
  1 RETCODE           (I4)     /* W/R: write/read
                                /* must be 0!
END-DEFINE
```

**Example:**

The following example writes the generated document to Standard Work File 10 and to PC Work File 15:

```
* -----
* CLASS NATURAL XML TOOLKIT - UTILITIES
*
* SDEMO_P1
*
* DESCRIPTION
* Serialize a given Data structure.
*
*
* AUTHOR SAG 01.2001
*
* VERSION 4.12.
*
* (c) Copyright Software AG 2001-2003. All rights reserved.
* -----
*
DEFINE DATA
LOCAL USING EMPL /* add generated data structure
LOCAL
1 XML (A) DYNAMIC
*
1 OUT (A72)
1 II (I4)
*
1 #CX (I4)
1 #CY (I4)
1 #CZ (I4)
1 FILENUMBER (I2)
1 OPERATION (A2)
1 RETCODE (I4)
END-DEFINE
/*[ initialize
EMPLOYEE.PERSONNEL-ID := 4711
*
EMPLOYEE.FIRST-NAME := "ADKINSON"
EMPLOYEE.NAME := "MARTHA"
*
EMPLOYEE.C$ADDRESS-LINE := 2
EMPLOYEE.ADDRESS-LINE(1) := "8603 GARLAND COURT"
EMPLOYEE.ADDRESS-LINE(2) := "FRAMINGHAM"
EMPLOYEE.ADDRESS-LINE(2) := "MA"
EMPLOYEE.CITY := "FRAMINGHAM"
EMPLOYEE.ZIP := "17010"
EMPLOYEE.COUNTRY := "USA"
*
EMPLOYEE.AREA-CODE := "617"
```

```

EMPLOYEE.PHONE := "210-4703"
*
EMPLOYEE.JOB-TITLE := "MANAGER"
EMPLOYEE.C$INCOME := 2
EMPLOYEE.SALARY(1) := 47000
EMPLOYEE.C$BONUS(1) := 2
EMPLOYEE.BONUS(1,1) := 10500
EMPLOYEE.BONUS(1,2) := 7875
*
EMPLOYEE.SALARY(2) := 47000
EMPLOYEE.C$BONUS(2) := 1
EMPLOYEE.BONUS(2,1) := 35700
*
INCLUDE EMPL-C "XML" "#CX" "#CY" "#CZ"
/* add generated Serialize
/* ]
*
ASSIGN FILENUMBER = 15 /* PC FILE
ASSIGN OPERATION = 'W'
CALLNAT 'XML2PCWR' XML FILENUMBER OPERATION RETCODE
PRINT XML
DEFINE WORK FILE 10 TYPE 'UNFORMATTED' /* STD WORK FILE
WRITE WORK FILE 10 VARIABLE XML
CLOSE WORK FILE 10
*
END

```

## Print Files

The XML Toolkit writes to Report 2.

## Invoking the Application

The XML toolkit is included in the library SYSEXXT.

### To use the XML Toolkit

1. In the Natural command line, enter LOGON SYSEXXT.
2. Enter XML2NAT.  
The Main Menu is displayed.

```

10:22:54          *** NATURAL XML Toolkit ***          2003-03-19
                  - Main Menu -                          Library SYSEXTC

Code  Function

      L  Generate from Natural Data Structure
      D  Generate from Document Type Definition
      O  Set up Specific Generation Options

Function Code .. _

Press PF4 or PF5 to start generation.
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      Help      Exit  LDA  DTD                                Optio  Canc

```

The following functions are available:

- Generate from Natural Data Structure  
Uses the Natural Data Area as a data source.
- Generate from Document Type Definition  
Uses the Document Type Definition (.dtd) as a data source.
- Set up Specific Generation Options

For details, refer to the corresponding sections.

## PF-Key Assignments

The following function keys are used for navigation and processing.

<b>PF1</b>	Help	Context-related help. For more information, refer to the online documentation.
<b>PF3</b>	Exit	On an options map: closes the function and saves the changes. On a generation map: closes the function after a generation is done. On the Main Menu map: closes the application.
<b>PF7</b>	Prev	Previous step (previous map).
<b>PF8</b>	Next	Next step (next map).
<b>PF9</b>	Finis(h)	Closes the function after a generation is done.
<b>PF12</b>	Cancel	Closes the function without saving the changes. Closes the function if no generation has already be done.

# Setting up Specific Generation Options

The generation options are arranged on two screens and are grouped into data fields and path definition.

The following topics are covered below:

- Invoking the Generation Options Setup Screens
- First Screen
- Second Screen

See also:

- Natural XML Toolkit Features
- Using a Natural Data Area as Data Source
- Using a Document Type Definition as Data Source
- PF-Key Assignments

## Invoking the Generation Options Setup Screens

▶ **To invoke the first screen of the generation options function**

- On the Main Menu screen, press PF10 **Optio(n)**.  
The first screen of the Options setup function is displayed.  
The map fields are described below.

### First Screen

Special characters that are not valid in XML have to be converted into valid names. The following map enables you to change the default conversion settings, if required.

```

09:51:35                *** NATURAL XML Toolkit ***                2003-03-20
User DEFAULT                - Options -                Library SYSEXXT

Additional fields
Counter separator character ..... @

XML name replacements
Namespace separator character ':' with .. $
Dot sign '.' with ..... /

Natural variable name replacements
Plus sign '+' with ..... plus_____
Hash / Number sign '#' with ..... hash_____
Slash sign '/' with ..... slash_____
At sign '@' with ..... at_____
Paragraph sign '§' with ..... paragraph__
Ampersand sign '&' with ..... ampersand__
Dollar sign '$' with ..... dollar_____

Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      Help           Exit                               Next           Canc

```

**Field Descriptions**

Counter Separator Character

<b>Belongs to Group:</b>	Additional Fields
<b>Format:</b>	A1
<b>Default Value:</b>	@

Namespace Separator Character ":" with:

<b>Belongs to Group:</b>	XML Name Replacements
<b>Format:</b>	A1
<b>Default Value:</b>	\$

Dot Sign '.' with:

<b>Belongs to Group:</b>	XML Name Replacements
<b>Format:</b>	A1
<b>Default Value:</b>	/

Plus Sign '+' with:

<b>Belongs to Group:</b>	XML Variable Name Replacements
<b>Format:</b>	A11
<b>Default Value:</b>	plus

Hash / Number Sign '#' with:

<b>Belongs to Group:</b>	Natural Variable Name Replacements
<b>Format:</b>	A11
<b>Default Value:</b>	hash

Slash Sign '/' with:

<b>Belongs to Group:</b>	Natural Variable Name Replacements
<b>Format:</b>	A11
<b>Default Value:</b>	slash

At Sign '@' with:

<b>Belongs to Group:</b>	Natural Variable Name Replacements
<b>Format:</b>	A11
<b>Default Value:</b>	at

Paragraph Sign '\$' with:

<b>Belongs to Group:</b>	Natural Variable Name Replacements
<b>Format:</b>	A11
<b>Default Value:</b>	para

Ampersand Sign '&' with:

<b>Belongs to Group:</b>	Natural Variable Name Replacements
<b>Format:</b>	A11
<b>Default Value:</b>	amp

Dollar Sign '\$' with:

<b>Belongs to Group:</b>	Natural Variable Name Replacements
<b>Format:</b>	A11
<b>Default Value:</b>	dollar

## Second Screen

The second map of the Options function serves to define the location of the target or source DTD file used for the conversion.

### To invoke the second screen of the Generation Options function

- On the Main Menu screen, press PF10 **Optio(n)**.
- On the first screen, press PF8 **Next**.  
The second screen of the Options function is displayed.  
The map fields are described below.

```

11:04:10          *** NATURAL XML Toolkit ***          2003-03-20
User DEFAULT          - Options -          Library SYSEXXT

External file:
/nat_64/proj/natc/511/samples/sysexxt/*.dtd_____

Natural library:
SYSEXXT_

Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      Help      Exit          Prev          Canc
    
```

**Field Descriptions**

External File

<b>Format:</b>	A253
<b>Default Value:</b>	(Resource directory of current library)

Natural Library

<b>Format:</b>	A8
<b>Default Value:</b>	(current library)

# Using a Natural Data Area as Data Source

This function enables you to generate an XML document from a data definition held in a Natural local, global or parameter data area.

The following topics are covered:

- Select Natural Data Area
- Select Root Group
- Generate Copycode for XML Parser Callback
- Generate Copycode for Serialization
- Generate File with DTD Definition
- Show Generation Results

See also:

- Using a Document Type Definition as Data Source
- Setting up Specific Generation Options
- PF-Key Assignments

## Select Natural Data Area

This screen serves to select LDA, GDA or PDA as input Data Area.

**Note:** The field entries shown in the screens below are default or example values.

```

08:07:48          ***** NATURAL XML Toolkit *****          2003-03-21
                - Generate from Natural Data Structure -   Library SYSEXXT

Select LDA, GDA or PDA as input Data Area.

Press 'Next' to read the Data Area.

Select a Level 1 group that should be used for further generations.

Select Data Area for generation
Library:  Type: Name:
SYSEXXT_ L   _____

Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      Help       Exit                Next                Canc
  
```

### Field Descriptions



<b>Format:</b>	A1
<b>Default Value:</b>	(All groups on Level 1)

Mark the desired element, e.g. EMPLOYEE, with an X and press ENTER.

## Generate Copycode for XML Parser Callback

This screen is used to generate copycode as implementation for the XML Parser Callback for the given group.

```

10:38:12          ***** NATURAL XML Toolkit *****          2003-03-21
          - Generate from Natural Data Structure -   Library SYSEXXT

Generate Copycode as implementation for the XML Parser Callback for the
given group.

Specify a Name and Press 'Next' to start the generation.

Press 'Next' to ignore this generation.

Select output for Parse Copycode
Library:  Type: Name:
SYSEXXT_  C   _____

Read Data Area done.
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      Help           Exit                   Prev  Next                               Canc
    
```

Generates the parser CALLBACK copycode. See also Parser CALLBACK Copycode (in the Examples document).

### Field Descriptions

Library

<b>Belongs to Group:</b>	Select Output Copycode
<b>Format:</b>	A8
<b>Default Value:</b>	(All libraries)

Type

<b>Belongs to Group:</b>	Select Output Copycode
<b>Format:</b>	A1
<b>Default Value:</b>	C - Copycode

Name

<b>Belongs to Group:</b>	Select Output Copycode
<b>Format:</b>	A8
<b>Default Value:</b>	(All objectsfs of the selected library and type)

Press PF8 Next to continue.

## Generate Copycode for Serialization

This screen is used to generate copycode as implementation for the serialization of the given group into a XML document.

```

10:56:06          ***** NATURAL XML Toolkit *****          2003-03-21
                    - Generate from Natural Data Structue -   Library SYSEXXT

Generate Copycode as implementation for the serialization of the given
group into a XML document.

Specify a Name and Press 'Next' to start the generation.

Press 'Next' to ignore this generation.

Select output for Serialize Copycode
Library:  Type:  Name:
SYSEXXT_  C    _____

Parse Copycode generation done.
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      Help      Exit      Prev Next      Canc
    
```

See also Serialize Copycode (in the Examples document).

### Field Descriptions

Library

<b>Belongs to Group:</b>	Select Output Copycode
<b>Format:</b>	A8
<b>Default Value:</b>	(All libraries)

Type

<b>Belongs to Group:</b>	Select Output Copycode
<b>Format:</b>	A21
<b>Default Value:</b>	Copycode

Name

<b>Belongs to Group:</b>	Select Output Copycode
<b>Format:</b>	A8
<b>Default Value:</b>	(All objects of the selected library and type)

Press PF8 Next to continue.

## Generate File with DTD Definition

This screen serves to generate a file with the DTD definition of a given group.

```

11:14:49          ***** NATURAL XML Toolkit *****          2003-03-21
          - Generate from Natural Data Structure -      Library SYSEXXT

Generate file with the dtd definition of a given group.

Specify a File Name and Press 'Next' to start the generation.

Press 'Next' to ignore this generation.

Select output for DTD file:
$NATDIR/$NATVERS/511/samples/sysexxt/*.dtd_____

Serialize Copycode generation done.
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      Help      Exit      Prev  Next      Canc
    
```

### Field Descriptions

Select Output for DTD File

<b>Format:</b>	A253
----------------	------

Press PF8 Next to continue.

## Show Generation Results

After the generation is complete, the generation results are displayed.

```

11:48:29          ***** NATURAL XML Toolkit *****          2003-03-21
          - Generate from Natural Data Structure -   Library SYSEXXT

Generation Results
Generate for Data Area
  Library ..: SYSEXXT
  Object ...: EMPL
  Read Data Area done.
Parser (Callback) Copycode
  Library ..: SYSEXXT
  Source ...: A1
  Parse Copycode generation done.
Serialize (Compress XML) Copycode
  Library ..: SYSEXXT
  Source ...: A2
  Serialize Copycode generation done.
File with dtd Definition
  File .....: $NATDIR/$NATVERS/511/samples/sysexxt/esi.dtd
  Generate DTD file.

Generation done.
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      Help      Exit              Prev      Finis      Canc

```

### Field Descriptions

#### Summary

<b>Format:</b>	A253/1:v
----------------	----------

Press PF9 **Finis**(h) to end the generation process.

# Using a Document Type Definition as Data Source

This function enables you to parse an XML document into a Natural variable defined in a local, global or parameter data area.

The following topics are covered:

- Generate From Document Type Definition
- Select Root Element
- Generate Copycode
- Serialize into XML Document
- Generate Natural Data Area
- Show Generation Results

See also:

- Using a Natural Data Area as Data Source
- Setting up Specific Generation Options
- PF-Key Assignments

## Generate From Document Type Definition

This screen is used to select a Document Type Definition/Tamino Schema as input Document Type.

**Note:** The field entries shown in the screens below are default or example values.

```

12:30:09          ***** NATURAL XML Toolkit *****          2003-03-21
                - Generate from Document Type Definition -   Library SYSEXXT

Select Document Type Definition/Tamino Schema as input Document Type.

Press 'Next' to read the Document Type.

Select a root element or Tamino Document Type that should be used
for further generations.

Select DTD/Tamino Schema for generation:
/nat_64/proj/natc/511/samples/sysexxt/empl.dtd_____

Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      Help           Exit           Next           Canc
  
```

**Field Descriptions****Input File**

<b>Format:</b>	A253
----------------	------

Use Tamino 2.1.x Schema instead of DTD.

Press PF8 Next to continue.

**Select Root Element**

This screen is used to select an element that should be the root of your XML document.

12:42:17	*** NATURAL XML Toolkit ***	2003-03-21
	- Select Root Element -	Library SYSEXXT
Element		
X	EMPLOYEE	
_	FULL-NAME	
_	FIRST-NAME	
_	NAME	
_	FULL-ADDRESS	
_	ADDRESS-LINE	
_	CITY	
_	ZIP	
_	COUNTRY	
_	TELEPHONE	
_	AREA-CODE	
_	PHONE	
_	JOB-TITLE	
_	INCOME	
_	SALARY	

**Field Descriptions**

Root Element

<b>Default Value:</b>	(All Elements)
-----------------------	----------------

Mark the desired element, e.g. EMPLOYEE, with an **X** and press ENTER.

**Generate Copycode**

This screen is used to generate copycode as implementation for the XML Parser Callback for the given group.

```

13:02:32          ***** NATURAL XML Toolkit *****          2003-03-21
          - Generate from Document Type Definition -   Library SYSEXXT

Generate Copycode as implementation for the XML Parser Callback for the
given group.

Specify a Name and Press 'Next' to start the generation.

Press 'Next' to ignore this generation.

Select output for Parse Copycode:
Library:  Type:  Name:
SYSEXXT  C

Read DTD/Tamino Schema done.
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      Help           Exit                               Prev  Next                               Canc
    
```

Generates the parser CALLBACK copycode. See also Parser CALLBACK Copycode (in the Examples document).

**Field Descriptions**

Library

<b>Belongs to Group:</b>	Select Output Copycode
<b>Format:</b>	A8
<b>Default Value:</b>	(All libraries)

Type

<b>Belongs to Group:</b>	Select Output Copycode
<b>Format:</b>	A1
<b>Default Value:</b>	Copycode

Name

<b>Belongs to Group:</b>	Select Output Copycode
<b>Format:</b>	A8
<b>Default Value:</b>	(All objects of the selected library and type)

Press PF8 Next to continue.

## Serialize into XML Document

This screen is used to generate copycode as implementation for the serialization of the given group into a XML document.

```

13:10:40          ***** NATURAL XML Toolkit *****          2003-03-21
          - Generate from Natural Data Structure -   Library SYSEXXT

Generate Copycode as implementation for the serialization of the given
group into a XML document.

Specify a Name and Press 'Next' to start the generation.

Press 'Next' to ignore this generation.

Select output for Serialize Copycode
Library:  Type: Name:
SYSEXXT_ C      A2_____

Parse Copycode generation done.
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      Help      Exit              Prev Next              Canc

```

See also Serialize Copycode (in the Examples document).

### Field Descriptions

Library

<b>Belongs to Group:</b>	Select Output Copycode
<b>Format:</b>	A8
<b>Default Value:</b>	(All libraries)

Type

<b>Belongs to Group:</b>	Select Output Copycode
<b>Format:</b>	A21
<b>Default Value:</b>	Copycode

Name

<b>Belongs to Group:</b>	Select Output Copycode
<b>Format:</b>	A8
<b>Default Value:</b>	(All objects of the selected library and type)

Press PF8 Next to continue.

## Generate Natural Data Area

This screen is used to generate a Natural Data Area with definition of a group that represents the XML document.

```

13:25:40          ***** NATURAL XML Toolkit *****          2003-03-21
                - Generate from Document Type Definition -   Library SYSEXXT

Generate Data Area with definition of a group that represents
the XML document.

Specify a Name and Press 'Next' to start the generation.

Press 'Next' to ignore this generation.

Select output LDA/GDA/PDA
Library:  Type: Name:
SYSEXXT_ L    A3_____

Generate Data Area.
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      Help      Exit                               Prev  Next                               Canc
    
```

### Field Descriptions

Library

<b>Belongs to Group:</b>	Select Output Copycode
<b>Format:</b>	A8
<b>Default Value:</b>	(All libraries)

Type

<b>Belongs to Group:</b>	Select Output Copycode
<b>Format:</b>	A21
<b>Default Value:</b>	L - Local Data Area

Name

<b>Belongs to Group:</b>	Select Output Copycode
<b>Format:</b>	A8
<b>Default Value:</b>	(All objects of the selected library and type)

Press PF8 Next to continue.

## Show Generation Results

After the generation is complete, the generation results summary is displayed.

```

13:51:11          ***** NATURAL XML Toolkit *****          2003-03-21
          - Generate from Document Type Definition -   Library SYSEXXT

Generation Results
Generate for DTD/ino schema
  File .....: /nat_64/proj/natc/511/samples/sysexxt/empl.dtd
  Read DTD/Tamino Schema done.
Parser (Callback) Copycode
  Library ..: SYSEXXT
  Source ...: A1
  Parse Copycode generation done.
Serialize (Compress XML) Copycode
  Library ..: SYSEXXT
  Source ...: A2
  Serialize Copycode generation done.
Data Area
  Library ..: SYSEXXT
  Source ...: A3
  Data Area Generation done.

Generation done.
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      Help           Exit                   Prev           Finis           Canc
  
```

### Field Descriptions

Summary

<b>Format:</b>	A253/1:v
----------------	----------

Press PF9 **Finis**(h) to end the generation process.

# Natural Simple XML Parser

The following topics are covered:

- Parser Description and Example
- Parser Restrictions

## Parser Description and Example

The Natural simple XML Parser enables you to parse XML documents with standard Natural programs. The parser sends an event, or runs an internal subroutine callback if the next part of the document is parsed. The inline subroutine "CALLBACK" is called with the name of the current element, text, comment within an xpath-like syntax. The parser engine is included as copy code "PARSER\_X". If an error occurs during parsing, e.g. the document is not wellformed, the "PARSER\_ERROR" inline subroutine is called and then the parser is canceled with "ESCAPE SUBROUTINE" (see also parser restrictions).

For extended error handling, it is possible to change the operand6 "Error Message Text" and operand7 "Error Number" to a value less than or equal to -9000. Then the "PARSER\_ERROR" inline subroutine is called and then the (sub)program is canceled with "ESCAPE SUBROUTINE". If other values are less than or equal to -8000, only the parser is canceled with "ESCAPE SUBROUTINE".

The major variables of the parser are defined at the Local Data Area "PARSER-X".

The parser copycode takes the following operands:

Operand	Format	Description																
1	A	XML file to be parsed																
2	A	ex-XPATH to represent element structure																
3	A1	Type of the XPATH content: <table border="1" data-bbox="418 1379 746 1733"> <thead> <tr> <th>Value</th> <th>Type</th> </tr> </thead> <tbody> <tr> <td>?</td> <td>Processing instruction</td> </tr> <tr> <td>D</td> <td>DOCTYPE</td> </tr> <tr> <td>!</td> <td>Comment</td> </tr> <tr> <td>C</td> <td>CDATA section</td> </tr> <tr> <td>T</td> <td>Starting Tag</td> </tr> <tr> <td>@</td> <td>Attribute</td> </tr> <tr> <td>/</td> <td>Close Tag</td> </tr> </tbody> </table>	Value	Type	?	Processing instruction	D	DOCTYPE	!	Comment	C	CDATA section	T	Starting Tag	@	Attribute	/	Close Tag
Value	Type																	
?	Processing instruction																	
D	DOCTYPE																	
!	Comment																	
C	CDATA section																	
T	Starting Tag																	
@	Attribute																	
/	Close Tag																	
4	A	Parsed Data																
5	L	Is TRUE if Parsed Data is empty																
6	A	Error Message Text																
7	I4	Error Number																

The XML file to be parsed (operand1) needs long alphanumeric or dynamic alphanumeric data. This data type is not available on the mainframe for the current Natural version. Therefore a second parser copycode "PARSER-MF" using an A1/1:v array as operand1 is delivered.

Return value of the XPATH data:

ex-Xpath	XML Structure
?	<? ... ?>
!DOCTYPE	<!DOCTYPE ... >
!DOCTYPE[	<!DOCTYPE .. [...]>
![CDATA[	<![CDATA[ ... ]]>
!--	<!-- -->
!	<! .. >
doc	<doc>
doc doc/foo doc/foo/\$ doc/foo// doc//	<doc><foo>text</foo></doc>
doc doc/@a1 doc//	<doc a1="a" />
doc doc/@a1 doc/@a2 doc/\$ doc//	<doc a1="a" a2="b">text</doc>
doc doc/\$ doc/foo doc/foo/\$ doc/foo// doc/\$ doc//	<doc> <foo>text</foo> </doc>
doc doc/![CDATA[ doc//	<doc><![CDATA[ ... ]]></doc>
doc doc/!-- doc//	<doc><!-- ... --></doc>

**Program Example:**

```

* -----
* CLASS NATURAL XML TOOLKIT - UTILITIES
*
*     PARSE
*
* DESCRIPTION
*     Parse given XML
*
*
* AUTHOR     SAG    01.2003
*

```

```

* VERSION          6.1.
*
* (c) Copyright Software AG 2003. All rights reserved.
*
* -----
*
DEFINE DATA LOCAL
1 XML_PARSER_INPUT          (A) DYNAMIC
1 XML_PARSER_ERROR_TEXT    (A253)
1 XML_PARSER_RESPONSE      (I4)
LOCAL USING PARSER-X        /* parser internal data - do not change
LOCAL
1 XML_PARSER_XPATH         (A) DYNAMIC
1 XML_PARSER_XPATH_TYPE    (A1)
1 XML_PARSER_CONTENT       (A) DYNAMIC
1 XML_PARSER_CONTENT_IS_EMPTY (L)
*
1 ANFANG                   (T)
* OUT                       (A) DYNAMIC
1 OUT                      (A126)
*
END-DEFINE
*
FORMAT (0) LS=128 PS=40
*
DEFINE WORK FILE 12 "E:\EMPLOYEE1.XML" TYPE "UNFORMATTED"
READ WORK FILE 12 XML_PARSER_INPUT
END-WORK
CLOSE WORK FILE 12
*
*
* ----- INCLUDE THE PARSER
INCLUDE PARSER_X 'XML_PARSER_INPUT' /* XML file to be parsed
'XML_PARSER_XPATH' /* XPATH to represent element...
'XML_PARSER_XPATH_TYPE' /* Type of callback
'XML_PARSER_CONTENT' /* Content of element found
'XML_PARSER_CONTENT_IS_EMPTY' /* Is TRUE if element is empty
'XML_PARSER_ERROR_TEXT' /* error Message
'XML_PARSER_RESPONSE' /* Error NR; 0 = OK
*
*
DEFINE SUBROUTINE CALLBACK
IF XML_PARSER_CONTENT_IS_EMPTY THEN
  IF XML_PARSER_XPATH_TYPE NE "T" AND XML_PARSER_XPATH_TYPE NE "/" THEN
    COMPRESS XML_PARSER_XPATH "(NULL)" INTO OUT WITH DELIMITER "="
  ELSE
    OUT := XML_PARSER_XPATH
  END-IF
ELSE
  COMPRESS XML_PARSER_XPATH XML_PARSER_CONTENT INTO OUT WITH DELIMITER "="
END-IF
WRITE OUT
END-SUBROUTINE
/*
DEFINE SUBROUTINE PARSER_ERROR
OUT := XML_PARSER_ERROR_TEXT
WRITE OUT
END-SUBROUTINE
END

```

With a given result document from Tamino for the Employee data, the result of this program looks like this:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<Employee xmlns:ino="http://namespaces.softwareag.com/tamino/response2" ino:id="560"
Personnel-ID="20006900">
  <Full-Name>
    <First-Name>JOE</First-Name>
    <Name>ATHERTON</Name>
  </Full-Name>
  <Mar-Stat>S</Mar-Stat>
  <Sex>M</Sex>
  <Birth>1941-02-21</Birth>
  <Full-Address>
    <Address-Line>11603 HUNTERS GREEN</Address-Line>
    <Address-Line>SYRACUSE</Address-Line>
    <Address-Line>NY</Address-Line>
    <City>SYRACUSE</City>
    <Zip>13201</Zip>
    <Post-Code>13201</Post-Code>
    <Country>USA</Country>
  </Full-Address>
  <Telephone>
    <Phone>173-9859</Phone>
    <Area-Code>315</Area-Code>
  </Telephone>
  <Dept>TECH10</Dept>
  <Job-Title>ANALYST</Job-Title>
  <Income>
    <Curr-Code>USD</Curr-Code>
    <Salary>43000</Salary>
  </Income>
  <Income>
    <Curr-Code>USD</Curr-Code>
    <Salary>39500</Salary>
  </Income>
  <Income>
    <Curr-Code>USD</Curr-Code>
    <Salary>36700</Salary>
  </Income>
  <Income>
    <Curr-Code>USD</Curr-Code>
    <Salary>34400</Salary>
  </Income>
  <Income>
    <Curr-Code>USD</Curr-Code>
    <Salary>32600</Salary>
  </Income>
  <Leave-Data>
    <Leave-Due>19</Leave-Due>
    <Leave-Taken>4</Leave-Taken>
  </Leave-Data>
  <Leave-Booked>
    <Leave-Start>19980112</Leave-Start>
    <Leave-End>19980112</Leave-End>
  </Leave-Booked>
  <Leave-Booked>
    <Leave-Start>19980605</Leave-Start>
    <Leave-End>19980605</Leave-End>
  </Leave-Booked>
  <Leave-Booked>
    <Leave-Start>19980916</Leave-Start>
```

```
<Leave-End>19980916</Leave-End>
</Leave-Booked>
<Lang>ENG</Lang>
</Employee>
```

**Note:** There is no line break in the whole document.

The result of the above Natural program looks like this:

```
?=xml version="1.0" encoding="ISO-8859-1"
Employee
Employee/@xmlns:ino=http://namespaces.softwareag.com/tamino/response2
Employee/@ino:id=560
Employee/@Personnel-ID=20006900
Employee/Full-Name
Employee/Full-Name/First-Name
Employee/Full-Name/First-Name/$=JOE
Employee/Full-Name/First-Name//
Employee/Full-Name/Name
Employee/Full-Name/Name/$=ATHERTON
Employee/Full-Name/Name//
Employee/Full-Name//
Employee/Mar-Stat
Employee/Mar-Stat/$=S
Employee/Mar-Stat//
Employee/Sex
Employee/Sex/$=M
Employee/Sex//
Employee/Birth
Employee/Birth/$=1941-02-21
Employee/Birth//
Employee/Full-Address
Employee/Full-Address/Address-Line
Employee/Full-Address/Address-Line/$=11603 HUNTERS GREEN
Employee/Full-Address/Address-Line//
Employee/Full-Address/Address-Line
Employee/Full-Address/Address-Line/$=SYRACUSE
Employee/Full-Address/Address-Line//
Employee/Full-Address/Address-Line
Employee/Full-Address/Address-Line/$=NY
Employee/Full-Address/Address-Line//
Employee/Full-Address/City
Employee/Full-Address/City/$=SYRACUSE
Employee/Full-Address/City//
Employee/Full-Address/Zip
Employee/Full-Address/Zip/$=13201
Employee/Full-Address/Zip//
Employee/Full-Address/Post-Code
Employee/Full-Address/Post-Code/$=13201
Employee/Full-Address/Post-Code//
Employee/Full-Address/Country
Employee/Full-Address/Country/$=USA
Employee/Full-Address/Country//
Employee/Full-Address//
Employee/Telephone
Employee/Telephone/Phone
Employee/Telephone/Phone/$=173-9859
Employee/Telephone/Phone//
Employee/Telephone/Area-Code
Employee/Telephone/Area-Code/$=315
Employee/Telephone/Area-Code//
Employee/Telephone//
```

```
Employee/Dept
Employee/Dept/$=TECH10
Employee/Dept//
Employee/Job-Title
Employee/Job-Title/$=ANALYST
Employee/Job-Title//
Employee/Income
Employee/Income/Curr-Code
Employee/Income/Curr-Code/$=USD
Employee/Income/Curr-Code//
Employee/Income/Salary
Employee/Income/Salary/$=43000
Employee/Income/Salary//
Employee/Income//
Employee/Income
Employee/Income/Curr-Code
Employee/Income/Curr-Code/$=USD
Employee/Income/Curr-Code//
Employee/Income/Salary
Employee/Income/Salary/$=39500
Employee/Income/Salary//
Employee/Income//
Employee/Income
Employee/Income/Curr-Code
Employee/Income/Curr-Code/$=USD
Employee/Income/Curr-Code//
Employee/Income/Salary
Employee/Income/Salary/$=36700
Employee/Income/Salary//
Employee/Income//
Employee/Income
Employee/Income/Curr-Code
Employee/Income/Curr-Code/$=USD
Employee/Income/Curr-Code//
Employee/Income/Salary
Employee/Income/Salary/$=34400
Employee/Income/Salary//
Employee/Income//
Employee/Income
Employee/Income/Curr-Code
Employee/Income/Curr-Code/$=USD
Employee/Income/Curr-Code//
Employee/Income/Salary
Employee/Income/Salary/$=32600
Employee/Income/Salary//
Employee/Income//
Employee/Leave-Data
Employee/Leave-Data/Leave-Due
Employee/Leave-Data/Leave-Due/$=19
Employee/Leave-Data/Leave-Due//
Employee/Leave-Data/Leave-Taken
Employee/Leave-Data/Leave-Taken/$=4
Employee/Leave-Data/Leave-Taken//
Employee/Leave-Data//
Employee/Leave-Booked
Employee/Leave-Booked/Leave-Start
Employee/Leave-Booked/Leave-Start/$=19980112
Employee/Leave-Booked/Leave-Start//
Employee/Leave-Booked/Leave-End
Employee/Leave-Booked/Leave-End/$=19980112
Employee/Leave-Booked/Leave-End//
Employee/Leave-Booked//
```

```

Employee/Leave-Booked
Employee/Leave-Booked/Leave-Start
Employee/Leave-Booked/Leave-Start/$=19980605
Employee/Leave-Booked/Leave-Start//
Employee/Leave-Booked/Leave-End
Employee/Leave-Booked/Leave-End/$=19980605
Employee/Leave-Booked/Leave-End//
Employee/Leave-Booked//
Employee/Leave-Booked
Employee/Leave-Booked/Leave-Start
Employee/Leave-Booked/Leave-Start/$=19980916
Employee/Leave-Booked/Leave-Start//
Employee/Leave-Booked/Leave-End
Employee/Leave-Booked/Leave-End/$=19980916
Employee/Leave-Booked/Leave-End//
Employee/Leave-Booked//
Employee/Lang
Employee/Lang/$=ENG
Employee/Lang//
Employee//

```

## Parser Restrictions

The parser does not handle:

- composition of a tag (incl. processing instruction). Only start-tag must be equal to end-tag. (incl. processing instruction).  
example:  

```
<.doc></.doc> <!-- invalid character in tag -->
<doc><? ?></doc> <!-- invalid whitespace -->
<doc>&#RE;</doc> <!-- invalid character in tag -->
```
- character or entity references  
example:  

```
<doc>& no refc</doc> <!-- missing semicolon -->
<doc al=v1></doc> <!-- string literal expected -->
```
- exact handling of CDATA-Sections  
example:  

```
<doc><![CDATA [ stuff]]></doc> <!-- must be CDATA[ -->
```
- content of an entity/processing instruction  
example:  

```
<doc>]]></doc> <!-- ]] not allowed -->
```
- number of tags/attributes
- headerinformation
- Unicode-charset (supports ISO-8859-1)

# Examples

The following examples are included:

- Serialize Copycode
- Generated Natural Data Area
- Natural DTD Parser
- Generated Type Definition
- Parser CALLBACK Copycode

## Serialize Copycode

Using the XML Toolkit, a copycode can be generated that can be used to convert a Natural group structure into an XML document.

The callback copycode takes following operands:

Operand	Format	Description	from PARSE- <b>X</b>																
1	A	ex-XPATH to represent element structure	operand2																
2	A1	Type of the XPATH content: <table border="1" style="margin-left: 20px;"> <thead> <tr> <th>Value</th> <th>Type</th> </tr> </thead> <tbody> <tr> <td>?</td> <td>Processing instruction</td> </tr> <tr> <td>D</td> <td>DOCTYPE</td> </tr> <tr> <td>!</td> <td>Comment</td> </tr> <tr> <td>C</td> <td>CDATA section</td> </tr> <tr> <td>T</td> <td>Starting Tag</td> </tr> <tr> <td>@</td> <td>Attribute</td> </tr> <tr> <td>/</td> <td>Close Tag</td> </tr> </tbody> </table>	Value	Type	?	Processing instruction	D	DOCTYPE	!	Comment	C	CDATA section	T	Starting Tag	@	Attribute	/	Close Tag	operand3
Value	Type																		
?	Processing instruction																		
D	DOCTYPE																		
!	Comment																		
C	CDATA section																		
T	Starting Tag																		
@	Attribute																		
/	Close Tag																		
3	A	Parsed Data	operand4																
4	L	Is TRUE if Parsed Data is empty	operand5																
5	I4	Counter Variable 1st Dimension																	
6	I4	Counter Variable 2nd Dimension																	
7	I4	Counter Variable 3rd Dimension																	

### Copycode Example EMPL-C:

```
* -----
* Parameter Definition
*
* &1& 'XML'                /* XML Document
* &2& '#CX'                 /* Counter Variable 1st Dimension
* &3& '#CY'                 /* Counter Variable 2nd Dimension
* &4& '#CZ'                 /* Counter Variable 3rd Dimension
* -----
* DTD E-\SAG\nat\NATAPPS\FUSER\XMLTK\RES\empl.dtd
COMPRESS &1& '<EMPLOYEE'
```

```

' PERSONNEL-ID="'EMPLOYEE.PERSONNEL-ID "'
'>' INTO &1& LEAVING NO
/* now the children
COMPRESS &1& '<FULL-NAME'
'>' INTO &1& LEAVING NO
/* now the children
COMPRESS &1& '<FIRST-NAME'
'>'
EMPLOYEE.FIRST-NAME
'</FIRST-NAME>' INTO &1& LEAVING NO
COMPRESS &1& '<NAME'
'>'
EMPLOYEE.NAME
'</NAME>' INTO &1& LEAVING NO
/*
COMPRESS &1& '</FULL-NAME>' INTO &1& LEAVING NO
COMPRESS &1& '<FULL-ADDRESS'
'>' INTO &1& LEAVING NO
/* now the children
FOR &2& = 1 TO EMPLOYEE.C@ADDRESS-LINE
COMPRESS &1& '<ADDRESS-LINE'
'>'
EMPLOYEE.ADDRESS-LINE(&2&)
'</ADDRESS-LINE>' INTO &1& LEAVING NO
END-FOR
COMPRESS &1& '<CITY'
'>'
EMPLOYEE.CITY
'</CITY>' INTO &1& LEAVING NO
COMPRESS &1& '<ZIP'
'>'
EMPLOYEE.ZIP
'</ZIP>' INTO &1& LEAVING NO
COMPRESS &1& '<COUNTRY'
'>'
EMPLOYEE.COUNTRY
'</COUNTRY>' INTO &1& LEAVING NO
/*
COMPRESS &1& '</FULL-ADDRESS>' INTO &1& LEAVING NO
COMPRESS &1& '<TELEPHONE'
'>' INTO &1& LEAVING NO
/* now the children
COMPRESS &1& '<PHONE'
'>'
EMPLOYEE.PHONE
'</PHONE>' INTO &1& LEAVING NO
COMPRESS &1& '<AREA-CODE'
'>'
EMPLOYEE.AREA-CODE
'</AREA-CODE>' INTO &1& LEAVING NO
/*
COMPRESS &1& '</TELEPHONE>' INTO &1& LEAVING NO
COMPRESS &1& '<JOB-TITLE'
'>'
EMPLOYEE.JOB-TITLE
'</JOB-TITLE>' INTO &1& LEAVING NO
FOR &2& = 1 TO EMPLOYEE.C@INCOME
COMPRESS &1& '<INCOME'
'>' INTO &1& LEAVING NO
/* now the children
COMPRESS &1& '<SALARY'
'>'

```

```

EMPLOYEE.SALARY(&2&)
'</SALARY>' INTO &1& LEAVING NO
FOR &3& = 1 TO EMPLOYEE.C@BONUS(&2&)
  COMPRESS &1& '<BONUS'
    '>'
    EMPLOYEE.BONUS(&2&,&3&)
    '</BONUS>' INTO &1& LEAVING NO
END-FOR
/*
COMPRESS &1& '</INCOME>' INTO &1& LEAVING NO
END-FOR
/*
COMPRESS &1& '</EMPLOYEE>' INTO &1& LEAVING NO

```

**Program Example:**

```

* -----
* CLASS NATURAL XML TOOLKIT
*
*
* DESCRIPTION
*           Serialize a given Data structure.
*
*
* AUTHOR           SAG    01.2003
*
* VERSION          6.1.
*
* (c) Copyright Software AG 2003. All rights reserved.
*
* -----
*
DEFINE DATA
LOCAL USING EMPL /* add generated data structure
LOCAL
1 XML              (A) DYNAMIC
*
1 OUT              (A72)
1 II               (I4)
*
1 OUTDYN (A) DYNAMIC
1 OBJLEN (I4)
1 OBJEND (I4)
1 OBJSTART (I4)
1 OBJLINE (I4)
*
1 #CX              (I4)
1 #CY              (I4)
1 #CZ              (I4)
END-DEFINE
*
EMPLOYEE.PERSONNEL-ID      := 4711
*
EMPLOYEE.FIRST-NAME       := "ADKINSON"
EMPLOYEE.NAME             := "MARTHA"
*
EMPLOYEE.C@ADDRESS-LINE   := 2
EMPLOYEE.ADDRESS-LINE(1) := "8603 GARLAND COURT"
EMPLOYEE.ADDRESS-LINE(2) := "FRAMINGHAM"
EMPLOYEE.ADDRESS-LINE(2) := "MA"
EMPLOYEE.CITY             := "FRAMINGHAM"

```

```

EMPLOYEE.ZIP           := "17010"
EMPLOYEE.COUNTRY      := "USA"
*
EMPLOYEE.AREA-CODE    := "617"
EMPLOYEE.PHONE       := "210-4703"
*
EMPLOYEE.JOB-TITLE    := "MANAGER"
EMPLOYEE.C@INCOME     := 2
EMPLOYEE.SALARY(1)   := 47000
EMPLOYEE.C@BONUS(1)  := 2
EMPLOYEE.BONUS(1,1)  := 10500
EMPLOYEE.BONUS(1,2)  := 7875
*
EMPLOYEE.SALARY(2)    := 47000
EMPLOYEE.C@BONUS(2)  := 1
EMPLOYEE.BONUS(2,1)  := 35700
*
INCLUDE EMPL-C "XML" "#CX" "#CY" "#CZ" /* add generated Serialize
*
FOR II = 1 TO *LENGTH(XML) STEP 72
  OUT := SUBSTR(XML,II)
  WRITE OUT
END-FOR
*
NEWPAGE
*
/* WRITE COMPLETE (A) DYNAMIC VARIABLE IF POSSIBLE USE CR AND IGNORE LF
OBJSTART := 1
*
EXAMINE xml FOR "><" REPLACE WITH ">" - H'0A' - "<"
EXAMINE xml FOR H'0A' GIVING POSITION OBJEND
*
REPEAT WHILE OBJEND NE 0
  /*
  IF OBJSTART GT 0 THEN
    ADD OBJSTART TO OBJEND
  END-IF
  /*
  OBJLEN := OBJEND - OBJSTART -1
  /*
  IF OBJLEN > 0 THEN
    OUTDYN := SUBSTRING(xml, OBJSTART, OBJLEN)
    /*
    FOR OBJLINE = 1 TO *LENGTH(OUTDYN) STEP 72
      OUT := SUBSTR (OUTDYN,OBJLINE)
      WRITE OUT
    END-FOR
  ELSE
    WRITE " "
  END-IF
  /*
  OBJSTART := OBJEND
  IF OBJSTART GT *LENGTH(xml)
    ESCAPE BOTTOM
  END-IF
  /*
  EXAMINE SUBSTRING(xml,OBJSTART) FOR H'0A' GIVING POSITION OBJEND
END-REPEAT
*
END

```

**Natural PDA EMPL Used:**

```

DEFINE DATA PARAMETER
1 EMPLOYEE
  2 ATTRIBUTES_OF_EMPLOYEE
    3 PERSONNEL-ID(A8)
*
  2 FULL-NAME
    3 FIRST-NAME(A20)
    3 NAME(A20)
*
  2 FULL-ADDRESS
    3 C@ADDRESS-LINE(I4)
    3 ADDRESS-LINE(A20/1:6)
    3 CITY(A20)
    3 ZIP(A20)
    3 COUNTRY(A3)
*
  2 TELEPHONE
    3 AREA-CODE(A6)
    3 PHONE(A15)
*
  2 JOB-TITLE(A25)
*
  2 C@INCOME(I4)
  2 INCOME(1:6)
    3 SALARY(A9)
    3 C@BONUS(I4)
    3 BONUS(A9/1:4)
END-DEFINE

```

## Generated Natural Data Area

Using the XML Toolkit, a Natural Data Area, or more precisely a Local Data Area, Parameter Data Area or Global Data Area, can be generated that represents a given Document Type Definition.

**Generation Rules:**

- Each Empty Element without Attributes (<!ELEMENT br EMPTY>) is generated as a Natural variable of Type B1. This is necessary, because empty Natural groups are not allowed.
- Each Empty Element with Attributes (<!ELEMENT br EMPTY><!ATTLIST br width CDATA #IMPLIED>) is generated as a Natural group.
- Each Element with content (<!ELEMENT b (#PCDATA)>) is generated as a Natural variable of type A253.
- Each Sequence of Elements (<!ELEMENT spec (front, body\*, back?)>) or Choice of Elements (<!ELEMENT div1 (p | list | note)>) is generated as a Natural group.
- Each clasped Sequence or Choice (<!ELEMENT address ( (street, housenumber), (zip, city) )>) is generated as a special group with the name prefix "###PSEUDO". This gives the possibility to represent the context or possible multiplicities.
- Each Attribute (<!ATTLIST br width CDATA #IMPLIED>) of an Element is generated as variable of Type A253 belonging to a group with the name prefix "ATTRIBUTES\_OF\_" followed by the name of the element.
- Multiple Elements are always generated as arrays of Dimension 1:v. The upper bound of the generated array has to be changed manually.
- If an Element is defined multiple (<!ELEMENT spec (front, body\*)>), an additional counter field C@BODY, is generated to specify the number of available elements.
- All names used inside the DTD are converted into upper case, because Natural names are not case sensitive. Duplicate names inside a generated group will be extended with a suffix to make the names unique.
- Special Characters not valid for Natural names are converted into valid Natural names. For the conversion

settings, see the option dialog of the XML Toolkit.

### Restrictions:

- Elements with Mixed content data (<!ELEMENT p (#PCDATA | a | ul | b | i | em)\*>) are not supported.
- DTDs that result in Natural data structures can not be used within Natural, because Natural only supports data structures with a maximum of three dimensions.

### Example DTD:

```
<!ELEMENT EMPLOYEE (FULL-NAME , FULL-ADDRESS , TELEPHONE ,JOB-TITLE, INCOME* )>
<!ATTLIST EMPLOYEE PERSONNEL-ID CDATA #REQUIRED >

<!ELEMENT FULL-NAME (FIRST-NAME , NAME )>
<!ELEMENT FIRST-NAME (#PCDATA )>
<!ELEMENT NAME (#PCDATA )>

<!ELEMENT FULL-ADDRESS (ADDRESS-LINE* , CITY , ZIP , COUNTRY )>
<!ELEMENT ADDRESS-LINE (#PCDATA )>
<!ELEMENT CITY (#PCDATA )>
<!ELEMENT ZIP (#PCDATA )>
<!ELEMENT COUNTRY (#PCDATA )>

<!ELEMENT TELEPHONE (PHONE , AREA-CODE )>
<!ELEMENT PHONE (#PCDATA )>
<!ELEMENT AREA-CODE (#PCDATA )>

<!ELEMENT JOB-TITLE (#PCDATA )>

<!ELEMENT INCOME (SALARY , BONUS* )>
<!ELEMENT SALARY (#PCDATA )>
<!ELEMENT BONUS (#PCDATA )>
```

### Generated Natural Data Area (*Italic* written parts of the DTD, but necessary for Natural):

```
DEFINE DATA PARAMETER
1 EMPLOYEE
  2 ATTRIBUTES_OF_EMPLOYEE
  3 PERSONNEL-ID(A253)
*
  2 FULL-NAME
  3 FIRST-NAME(A253)
  3 NAME(A253)
*
  2 FULL-ADDRESS
  3 C@ADDRESS-LINE(I4)
  3 ADDRESS-LINE(A253/1:v)
  3 CITY(A253)
  3 ZIP(A253)
  3 COUNTRY(A253)
*
  2 TELEPHONE
  3 AREA-CODE(A253)
  3 PHONE(A253)
*
  2 JOB-TITLE(A253)
*
  2 C@INCOME(I4)
  2 INCOME(1:v)
```

```

3 SALARY(A253)
3 C@BONUS(I4)
3 BONUS(A253/1:v)
END-DEFINE

```

## Natural DTD Parser

### Translation Rules:

Natural	Document Type Definiton
1 G1 2 E1 (A )	<!ELEMENT G1 (E1)> <!ELEMENT E1 (#PCDATA)>
1 G1 2 E1 (A ) 2 E2 (A ) 2 E3 (A )	<!ELEMENT G1 (E1, E2, E3)> <!ELEMENT E1 (#PCDATA)> <!ELEMENT E2 (#PCDATA)> <!ELEMENT E3 (#PCDATA)>
1 C@E1_MAX (I4) CONST <10> 1 G1 2 C@E1 (I4) 2 E1 (A /1:C@E1_MAX)	<!ELEMENT G1 (E1*)> <!ELEMENT E1 (#PCDATA)>
1 C@E1_MAX (I4) CONST <10> 1 G1 2 C@E1 (I4) 2 E1 (A /1:C@E1_MAX)	<!ELEMENT G1 (E1+)> <!ELEMENT E1 (#PCDATA)>
1 G1 2 E1 (A )	<!ELEMENT G1 (E1?)> <!ELEMENT E1 (#PCDATA)>
1 G1 2 E1 (A ) 2 E2 (A ) 2 E3 (A )	<!ELEMENT G1 (E1  E2  E3)> <!ELEMENT E1 (#PCDATA)> <!ELEMENT E2 (#PCDATA)> <!ELEMENT E3 (#PCDATA)>
1 G1 2 E1 (A ) 2 E2 (A ) 2 G2 2 E1_2 (A ) 2 E3 (A )	<!ELEMENT G1 (E1, E2, G2)> <!ELEMENT E1 (#PCDATA)> <!ELEMENT E2 (#PCDATA)> <!ELEMENT G2 (E1, E3)> <!ELEMENT E3 (#PCDATA)>
1 #G1 2 #E1 (A )	<!ELEMENT G1 (E1)> <!ELEMENT E1 (#PCDATA)>
2 E1 (A ) 3 ATTRIBUTES_OF_E1 4 A1 (A ) CONST <'schema'> 4 A2 (A ) 4 A3 (A )	<!ELEMENT E1 (#PCDATA)> <!ATTLIST E1 A1 #FIXED "schema" A2 NMTOKEN #IMPLIED A3 ID #REQUIRED>

## Generated Type Definition

Using the XML Toolkit, a Natural Data Area, or more precisely a Local Data Area, Parameter Data Area or Global Data Area, can be generated that represents a given Document Type Definition.

**Generation Rules:**

- Natural Variable will result in an Element with content.
- Natural Group will result in a Sequence of Elements.
- Multiple Variables or Groups will be generated with multiplicity "zero or more".
- Special Characters not valid for xml names are converted into valid names. For the conversion settings, see the options dialog of the XML Toolkit.

**Example Natural Data Area:**

```

DEFINE DATA LOCAL
1 NAT$EMPLOYEE
  2 ATTRIBUTES_OF_NAT$EMPLOYEE
    3 PERSONNEL/ID(A8)
  2 C@MAN@WORK(I4)
  2 MAN@WORK
    3 JOB(A10)
  2 A$TEST$MAKL(I4)
  2 AS/FA/SD(P7.5)
  2 #ASDFAS(F4)
  2 ASF#AS(N9)
  2 A-SF-D(A) Dynamic
  2 INC@OME(1:6)
    3 C@BONUS(I4)
    3 BONUS(A9/1:4)
END-DEFINE

```

**Generated DTD:**

```

<!-- DTD XMLTOOLS BEISP -->
<!ELEMENT NATdollarEMPLOYEE ( MANatWORK , AdollarTESTdollarMAKL ,
    ASslashFAslashSD , hashASDFAS , ASFhashAS , A-SF-D , INCatOME* ) >
<!ATTLIST NATdollarEMPLOYEE PERSONNELslashID CDATA #IMPLIED >
<!ELEMENT MANatWORK ( JOB ) >
<!ELEMENT JOB (#PCDATA) >
<!ELEMENT AdollarTESTdollarMAKL (#PCDATA) >
<!ELEMENT ASslashFAslashSD (#PCDATA) >
<!ELEMENT hashASDFAS (#PCDATA) >
<!ELEMENT ASFhashAS (#PCDATA) >
<!ELEMENT A-SF-D (#PCDATA) >
<!ELEMENT INCatOME ( BONUS* ) >
<!ELEMENT BONUS (#PCDATA) >

```

## Parser CALLBACK Copycode

Using the XML Toolkit, a copycode can be generated that can be used with the Natural Simple XML Parser.

The callback copycode takes the following operands:

Operand	Format	Description	from PARSE-X																
1	A	ex-XPATH to represent element structure	operand2																
2	A1	Type of the XPATH content: <table border="1" data-bbox="416 450 746 801"> <thead> <tr> <th>Value</th> <th>Type</th> </tr> </thead> <tbody> <tr> <td>?</td> <td>Processing instruction</td> </tr> <tr> <td>D</td> <td>DOCTYPE</td> </tr> <tr> <td>!</td> <td>Comment</td> </tr> <tr> <td>C</td> <td>CDATA section</td> </tr> <tr> <td>T</td> <td>Starting Tag</td> </tr> <tr> <td>@</td> <td>Attribute</td> </tr> <tr> <td>/</td> <td>Close Tag</td> </tr> </tbody> </table>	Value	Type	?	Processing instruction	D	DOCTYPE	!	Comment	C	CDATA section	T	Starting Tag	@	Attribute	/	Close Tag	operand3
Value	Type																		
?	Processing instruction																		
D	DOCTYPE																		
!	Comment																		
C	CDATA section																		
T	Starting Tag																		
@	Attribute																		
/	Close Tag																		
3	A	Content of found element	operand4																
4	L	Is TRUE if Parsed Data is empty	operand5																
5	I4	Counter Variable 1st Dimension																	
6	I4	Counter Variable 2nd Dimension																	
7	I4	Counter Variable 3rd Dimension																	

**Copycode Example EMPL-P:**

```

* -----
* Parameter Definition
*
* &1& 'XML_PARSER_XPATH'           /* XPATH to represent element...
* &2& 'XML_PARSER_XPATH_TYPE'     /* Type of the XPATH:
*                                ? Processing instruction
*                                D DOCTYPE
*                                ! Comment
*                                C CDATA section
*                                T Starting Tag
*                                @ Attribute
*                                / Close Tag
*                                $ Parsed Data
* &3& 'XML_PARSER_CONTENT'        /* Content of found element
* &4& 'XML_PARSER_CONTENT_IS_EMPTY' /* Is TRUE if Content is empty
* &5& '#CX'                        /* Counter Variable 1st Dimension
* &6& '#CY'                        /* Counter Variable 2nd Dimension
* &7& '#CZ'                        /* Counter Variable 3rd Dimension
* -----
*
DECIDE ON FIRST &1&
VALUE 'EMPLOYEE'
RESET EMPLOYEE
VALUE 'EMPLOYEE/@PERSONNEL-ID'
/* #REQUIRED
EMPLOYEE.PERSONNEL-ID := &3&
VALUE 'EMPLOYEE/FULL-NAME'
IGNORE
VALUE 'EMPLOYEE/FULL-NAME/FIRST-NAME'
IGNORE
VALUE 'EMPLOYEE/FULL-NAME/FIRST-NAME/$'
EMPLOYEE.FIRST-NAME := &3&

```

```

VALUE 'EMPLOYEE/FULL-NAME/NAME'
  IGNORE
VALUE 'EMPLOYEE/FULL-NAME/NAME/$'
  EMPLOYEE.NAME := &3&
VALUE 'EMPLOYEE/FULL-ADDRESS'
  IGNORE
VALUE 'EMPLOYEE/FULL-ADDRESS/ADDRESS-LINE'
  /* OPTIONAL MULTIPLE IST: 18 PARENT: FULL-ADDRESS
  ADD 1 TO EMPLOYEE.C@ADDRESS-LINE
VALUE 'EMPLOYEE/FULL-ADDRESS/ADDRESS-LINE/$'
  &5& := EMPLOYEE.C@ADDRESS-LINE
  EMPLOYEE.ADDRESS-LINE(&5&) := &3&
VALUE 'EMPLOYEE/FULL-ADDRESS/CITY'
  IGNORE
VALUE 'EMPLOYEE/FULL-ADDRESS/CITY/$'
  EMPLOYEE.CITY := &3&
VALUE 'EMPLOYEE/FULL-ADDRESS/ZIP'
  IGNORE
VALUE 'EMPLOYEE/FULL-ADDRESS/ZIP/$'
  EMPLOYEE.ZIP := &3&
VALUE 'EMPLOYEE/FULL-ADDRESS/COUNTRY'
  IGNORE
VALUE 'EMPLOYEE/FULL-ADDRESS/COUNTRY/$'
  EMPLOYEE.COUNTRY := &3&
VALUE 'EMPLOYEE/TELEPHONE'
  IGNORE
VALUE 'EMPLOYEE/TELEPHONE/PHONE'
  IGNORE
VALUE 'EMPLOYEE/TELEPHONE/PHONE/$'
  EMPLOYEE.PHONE := &3&
VALUE 'EMPLOYEE/TELEPHONE/AREA-CODE'
  IGNORE
VALUE 'EMPLOYEE/TELEPHONE/AREA-CODE/$'
  EMPLOYEE.AREA-CODE := &3&
VALUE 'EMPLOYEE/JOB-TITLE'
  IGNORE
VALUE 'EMPLOYEE/JOB-TITLE/$'
  EMPLOYEE.JOB-TITLE := &3&
VALUE 'EMPLOYEE/INCOME'
  /* OPTIONAL MULTIPLE IST: 18 PARENT: EMPLOYEE
  ADD 1 TO EMPLOYEE.C@INCOME
VALUE 'EMPLOYEE/INCOME/SALARY'
  IGNORE
VALUE 'EMPLOYEE/INCOME/SALARY/$'
  &5& := EMPLOYEE.C@INCOME
  EMPLOYEE.SALARY(&5&) := &3&
VALUE 'EMPLOYEE/INCOME/BONUS'
  /* OPTIONAL MULTIPLE IST: 18 PARENT: INCOME
  &5& := EMPLOYEE.C@INCOME
  ADD 1 TO EMPLOYEE.C@BONUS(&5&)
VALUE 'EMPLOYEE/INCOME/BONUS/$'
  &5& := EMPLOYEE.C@INCOME
  &6& := EMPLOYEE.C@BONUS(&5&)
  EMPLOYEE.BONUS(&5&,&6&) := &3&
NONE
IGNORE
END-DECIDE

```

**Subprogram Example:**

```

* -----
* CLASS NATURAL XML TOOLKIT - UTILITIES
*
*
* DESCRIPTION
*         Parse a given XML document.
*
*
* AUTHOR      SAG    01.2003
*
* VERSION     6.1.
*
* (c) Copyright Software AG 2003. All rights reserved.
*
* -----
*
DEFINE DATA PARAMETER
1 XML_PARSER_INPUT           (A) DYNAMIC
PARAMETER USING EMPL
PARAMETER
1 XML_PARSER_ERROR_TEXT     (A253)
1 XML_PARSER_RESPONSE       (I2)
*
LOCAL USING PARSER-X
LOCAL
1 XML_PARSER_XPATH         (A) DYNAMIC
1 XML_PARSER_XPATH_TYPE    (A1)
1 XML_PARSER_CONTENT       (A) DYNAMIC
1 XML_PARSER_CONTENT_IS_EMPTY (L)
*
LOCAL
1 #CX                      (I4)
1 #CY                      (I4)
1 #CZ                      (I4)
END-DEFINE
*
* ----- INCLUDE THE PARSER
INCLUDE PARSER_X 'XML_PARSER_INPUT' /* XML file to be parsed
'XML_PARSER_XPATH' /* XPATH to represent element...
'XML_PARSER_XPATH_TYPE' /* Type of callback
'XML_PARSER_CONTENT' /* Content of found element
'XML_PARSER_CONTENT_IS_EMPTY' /* Is TRUE if element is empty
'XML_PARSER_ERROR_TEXT' /* error Message
'XML_PARSER_RESPONSE' /* Error NR; 0 = OK
*
* ----- CALLBACK HANDLER
DEFINE SUBROUTINE CALLBACK
*
INCLUDE EMPL-P 'XML_PARSER_XPATH' /* XPATH to represent element...
'XML_PARSER_XPATH_TYPE' /* Type of callback
'XML_PARSER_CONTENT' /* Content of found element
'XML_PARSER_CONTENT_IS_EMPTY' /* Is TRUE if element is empty
'#CX'
'#CY'
'#CZ'
*
END-SUBROUTINE
/*
DEFINE SUBROUTINE PARSER_ERROR
IGNORE
END-SUBROUTINE
END

```

**Natural PDA EMPL Used:**

```
DEFINE DATA PARAMETER
1 EMPLOYEE
  2 ATTRIBUTES_OF_EMPLOYEE
    3 PERSONNEL-ID(A8)
  *
  2 FULL-NAME
    3 FIRST-NAME(A20)
    3 NAME(A20)
  *
  2 FULL-ADDRESS
    3 C@ADDRESS-LINE(I4)
    3 ADDRESS-LINE(A20/1:6)
    3 CITY(A20)
    3 ZIP(A20)
    3 COUNTRY(A3)
  *
  2 TELEPHONE
    3 AREA-CODE(A6)
    3 PHONE(A15)
  *
  2 JOB-TITLE(A25)
  *
  2 C@INCOME(I4)
  2 INCOME(1:6)
    3 SALARY(A9)
    3 C@BONUS(I4)
    3 BONUS(A9/1:4)
END-DEFINE
```

# Parser Error Messages

The following error messages will be produced by the parser:

Response	Error Text	Example
00	Parse ended without errors.	valid/*
-01	Wrong character set/Document does not start with '<'.	not-wf/sa/147.xml
-02	Processing instruction was not closed. Position %2%.	not-wf/sa/004.xml
-03	A CDATA section was not closed. Position %2%.	esi/001.xml
-04	!DOCTYPE section was not closed. Position %2%.	not-wf/sa/055.xml
-05	Incorrect syntax was used in a comment. Position %2%.	not-wf/sa/006.xml
-06	A comment was not closed. Position %2%.	not-wf/sa/027.xml
-07	A CDATA section was not closed. Position %2%	not-wf/sa/017.xml
-08	A comment section was not closed. Position %2%.	esi/002.xml
-09	Closing tag name was started with an invalid character. Position %2%.	not-wf/sa/019.xml
-10	Closing tag without starting element. Position %2%.	not-wf/sa/042.xml
-11	Closing tag '%3%' does not match the start tag '%1%'. Position %2%.	not-wf/sa/039.xml
-12	Closing tag was not closed. Position %2%.	esi/003.xml
-13	Closing tag '%1%' was not closed. Position %2%.	
-14	Starting tag name was started with an invalid character. Position %2%.	not-wf/sa/035.xml
-15	Attribute name of tag '%1%' not found. Position %2%.	
-16	Attribute name of tag '%1%' contains an invalid character. Position %2%.	not-wf/sa/001.xml
-17	Attribute value of tag '%1%' ending quotation mark missing. Position %2%.	not-wf/sa/013.xml
-18	Attribute value of tag '%1%' ending apostrophe missing. Position %2%.	esi/005.xml
-19	Starting tag section was not closed. Position %2%.	esi/006.xml
-20	Tag '%1%' section was not closed. Position %2%.	not-wf/sa/176.xml
-21	A section was not closed. Position %2%.	not-wf/sa/025.xml
< -8000	User defined error messages, parser ends.	
< -9000	User defined error messages, PARSE_ERROR is called and parser ends.	